An Extended View: ReLU Networks as Gated Linear Systems and Associative Memories

1. Start from the Exact Identity

At the unit level, everything follows from one simple equivalence:

- A ReLU is not just a nonlinear function
- It is a **product of a binary decision and a continuous value**

This can be read as:

- a **decision process**: is the unit active?
- a **signal path**: what value flows if it is?

From this alone, a different systems picture emerges.

---

2. From Units to Layers: Structured Gating

Across a layer:

- Each neuron produces a binary decision (on/off)
- These decisions form a **gating vector**
- That vector determines which computations are active

So a layer does two things simultaneously:

1. **Partitions the input space** (via hyperplanes)
2. **Selects a subset of linear components**

This makes the layer:

> a *data-dependent routing mechanism* over a shared pool of linear features

---

3. From Layers to Networks: Exponential Model Families

When you stack layers:

- Each layer refines the gating pattern

- The number of possible activation patterns grows combinatorially

For any fixed input:

- all gates are fixed
- the network collapses to a **single linear map**

Across inputs:

- different gating patterns → different linear maps

So the network is:

> a very large collection of linear models, with **shared parameters**, selected by input-dependent gating

This connects directly to ideas already familiar in:

- conditional computation
- sparse mixture-of-experts
- routing networks

The difference is granularity: ReLU implements this **implicitly at every neuron**.

---

4. Associative Memory Interpretation

With this structure in place, a natural interpretation emerges:

- The **activation pattern** is an *address*
- The **weights** are *stored content*
- The output is a **retrieval conditioned on that address**

Each layer:

- maps inputs → addresses
- uses those addresses to retrieve and combine features

Stacked layers:

- progressively refine the address
- increase selectivity and specificity

So the network behaves like:

> a hierarchy of **content-addressable systems with learned addressing**

---

5. Locality: Why It Generalizes

Because gating is defined by hyperplanes:

- Nearby inputs tend to lie on the same side of most hyperplanes
- → they share most of their gating pattern

This gives:

- **local stability** (same linear model in a region)
- **controlled transitions** (only a few gates flip across boundaries)

This is closely aligned with **locality-sensitive hashing**:

- Similar inputs → similar codes
- Dissimilar inputs → different codes

But here:

> the hash is *learned*, not fixed

---

6. Capacity and Interference

From this viewpoint, capacity is governed by:

- how many distinct activation patterns are effectively used
- how much those patterns **overlap in their use of weights**

Two regimes emerge:

Low overlap

- Different inputs activate different subsets
- Behavior:
    - clean associations
    - stable outputs

High overlap

- Many inputs reuse the same subsets
- Behavior:

    - interference
    - noise-like errors (aggregation of many small contributions)

This mirrors classical associative memory behavior:

- superposition → Gaussian-like noise
- limited capacity → crosstalk

---

7. Training as Co-Adaptation

Training can be viewed as two coupled processes:

(A) Adjusting the gates (geometry)

- Moving hyperplanes
- Changing which inputs share activation patterns

(B) Adjusting the content (weights)

- Improving outputs given the current gating

This creates a feedback loop:

1. Current gating defines a partition of the data
2. Weights learn best possible mappings within that partition
3. Gradients push the partition to reduce conflicts

So training is not just fitting a function—it is:

> **learning how to partition the input space so that simple linear models suffice within each region**

---

8. A More Operational Perspective

To make this viewpoint concrete, a few measurable quantities are useful:

- **Activation pattern diversity**
    - How many distinct patterns occur?
- **Pattern reuse**
    - How many inputs share the same pattern?
- **Stability**

    - How often do small perturbations flip gates?
- **Overlap structure**
    - How many units are commonly shared between different inputs?
- **Region conditioning**
    - For a fixed pattern, how stable is the induced linear map?

These directly reflect:

- capacity
- generalization
- robustness

---

9. Relation to Known Architectures

This framing aligns naturally with:

- **Mixture of Experts**
    - explicit gating over experts
- **Sparse routing models**
    - input-dependent computation paths
- **Random feature models**
    - fixed projections + learned readout

ReLU networks can be seen as:

> a dense, fine-grained, implicitly gated version of these systems

---

10. A Practical Mental Model

A compact way to hold all of this:

> A ReLU network is a system that
> **(1) computes a binary code for the input, and**
> **(2) uses that code to select and combine linear components.**

Everything else—depth, width, optimization—operates within this structure.

---

11. Why This View Can Be Useful

This perspective does not replace existing frameworks, but it can help clarify:

- why piecewise linear models scale so well
- how overparameterization enables reuse
- where instability comes from (gate flipping)
- how generalization depends on region geometry

It also suggests a more explicit methodology:

- analyze **gating structure** and **linear behavior** separately
- treat networks as **ensembles of linear systems indexed by activation patterns**

Closing Thought

The central point is simple but far-reaching:

> **ReLU introduces discrete structure (gating) inside a continuous model.**

Once that is taken literally—not just metaphorically—the network becomes easier to interpret as:

- a routing system,
- a learned hashing scheme,
- and a layered associative memory.

All three views describe the same mechanism from different angles, and together they provide a more complete picture of what these models are doing.